New syllabus 2023-24

Chapter 7

Control Statements

Informatics Practices

Class XI (As per CBSE Board)

Control Statements

Control statements are used to control the flow of execution depending upon the specified condition/logic.

There are three types of control statements.

1. Decision Making Statements
2. Iteration Statements (Loops)
3. Jump Statements (break, continue, pass)

Decision Making Statement

Decision making statement used to control the flow of execution of program depending upon condition.

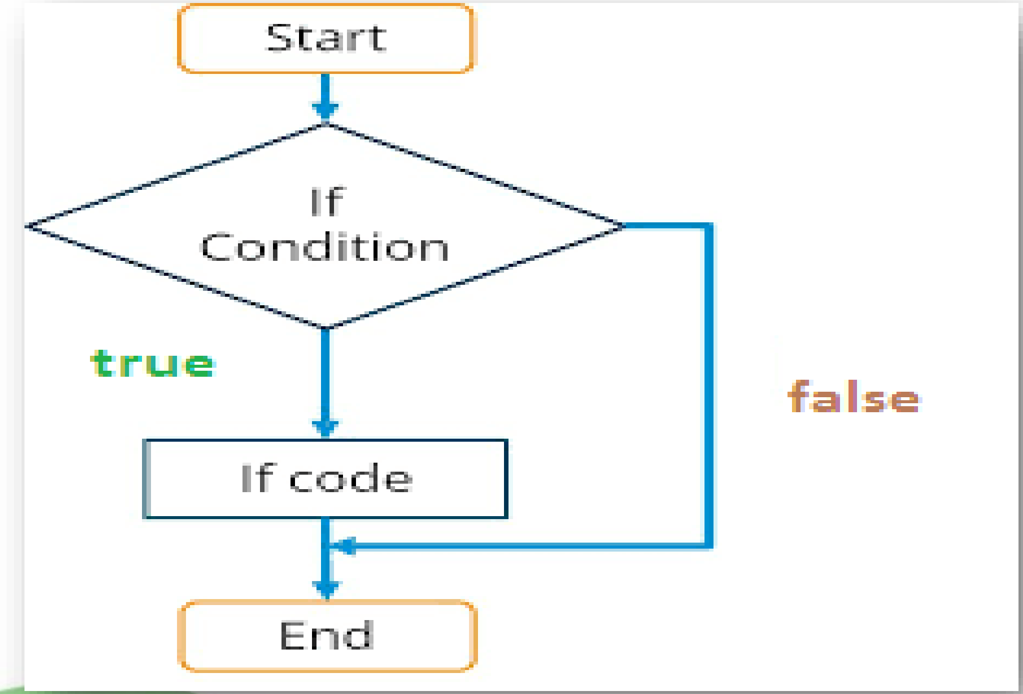
There are three types of decision making statement.

1. if statements
2. if-else statements
3. Nested if-else statement

Decision Making Statement

1. if statements

An if statement is a programming conditional statement that, if proved true, performs a function or displays information.



Decision Making Statement

1. if statements

Syntax:

```
if(condition):
        statement
        [statements]
```

e.g.

```
noofbooks = 2
if (noofbooks == 2):
        print('You have ')
        print(‘two books’)
print(‘outside of if statement’)
```

Output

You have two books

Note:To indicate a block of code in Python, you must indent each line of the block by the same amount. In above e.g. both print statements are part of if condition because of both are at same level indented but not the third print statement.

Decision Making Statement

1. if statements

Using logical operator in if statement

```
x=1
y=2
if(x==1 and y==2):
  print('condition matcing the criteria')
```

Output :-

condition matcing the criteria

```
a=100
if not(a == 20):
  print('a is not equal to 20')
```

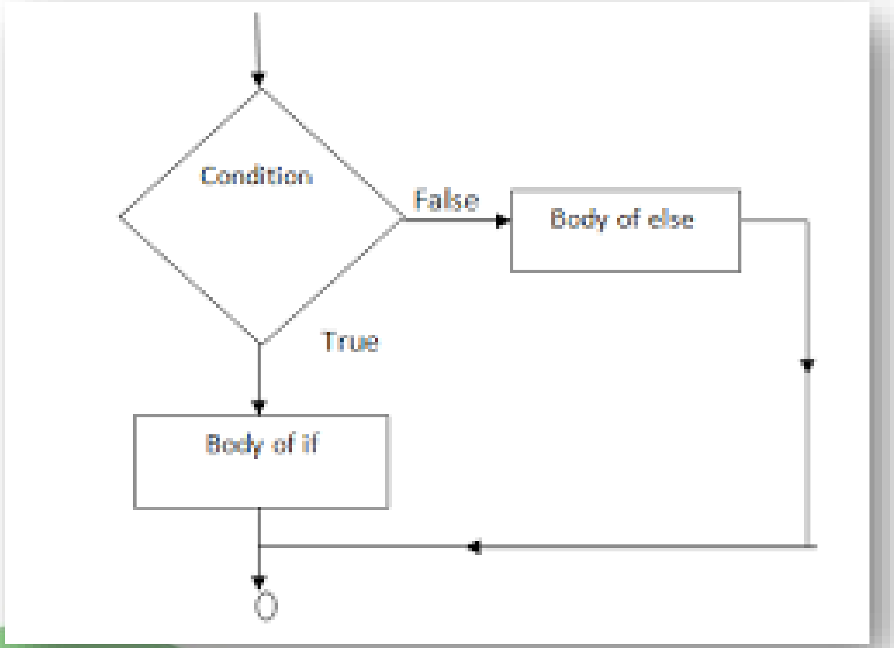
Output :-

a is not equal to 20

Decision Making Statement

2. if-else Statements

If-else statement executes some code if the test expression is true (nonzero) and some other code if the test expression is false.



Decision Making Statement

2. if-else Statements

Syntax:

```
        if(condition):
                statements
        else:
                statements
```

e.g.

```
a=10
if(a < 100):
  print(‘less than 100')
else:
  print(‘more than equal 100')
```

OUTPUT

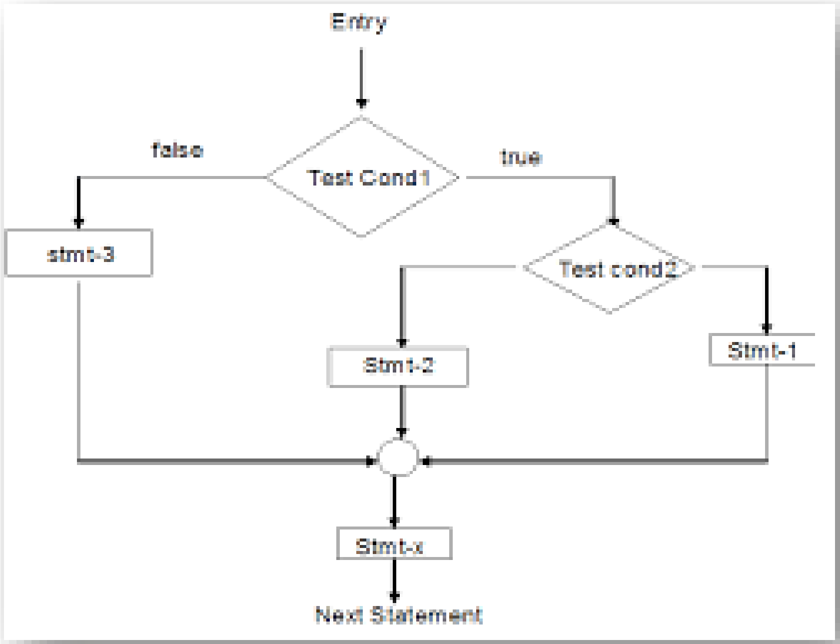
less than 100

*Write a program in python to check that entered numer is even or odd

Decision Making Statement

3. Nested if-else statement

The nested if...else statement allows you to check for multiple test expressions and execute different codes for more than two conditions.



3. Nested if-else statement

Syntax

```
If (condition):
        statements
elif (condition):
        statements
else:
        statements
```

E.G.

```
num = float(input("Enter a number: "))
if num >= 0:
   if num == 0:
      print("Zero")
   else:
      print("Positive number")
else:
   print("Negative number")
```

OUTPUT

Enter a number: 5

Positive number

* Write python program to find out largest of 3 numbers.

Iteration Statements (Loops)

Iteration statements(loop) are used to execute a block of statements as long as the condition is true.
Loops statements are used when we need to run same code again and again.

Python Iteration (Loops) statements are of three type :-

1. While Loop

2. For Loop

3. Nested For Loops

Iteration Statements (Loops)

1. While Loop

It is used to execute a block of statement as long as a given condition is true. And when the condition become false, the control will come out of the loop. The condition is checked every time at the beginning of the loop.

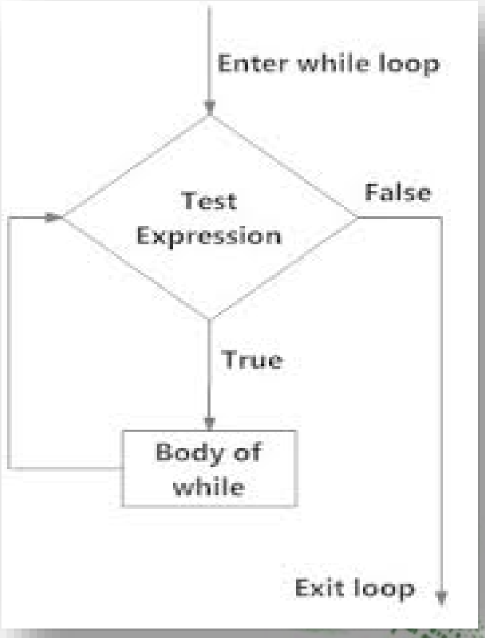
Syntax

```
while (condition):
        statement
        [statements]
```

e.g.

```
x = 1
while (x <= 4):
  print(x)
  x = x + 1
```

Output
1
2
3
4



Iteration Statements (Loops)

While Loop continue

While Loop With Else

e.g.

```
x = 1
while (x < 3):
  print('inside while loop value of x is ',x)
  x = x + 1
else:
  print('inside else value of x is ', x)
```

Output
inside while loop value of x is 1
inside while loop value of x is 2
inside else value of x is 3

***Write a program in python to find out the factorial of a given number**

Iteration Statements (Loops)

While Loop continue

Infinite While Loop

e.g.

```
x = 5
while (x == 5):
  print('inside loop')
```

Output

Inside loop

Inside loop

…

…

Iteration Statements (Loops)

2. For Loop

It is used to iterate over items of any sequence, such as a list or a string.

Syntax

```
for val in sequence:
        statements
```

e.g.

```
for i in range(3,5):
  print(i)
```

Output

3

4

Iteration Statements (Loops)

2. For Loop continue

Example programs

```
for i in range(5,3,-1):
  print(i)
```

Output

5

4

range() Function Parameters

start: Starting number of the sequence.

stop: Generate numbers up to, but not including this number.

step(Optional): Determines the increment between each numbers in the sequence.

Iteration Statements (Loops)

2. For Loop continue

For Loop With Else

e.g.

```
for i in range(1, 4):
   print(i)
else:  # Executed because no break in for
   print("No Break")
```

Output

```
1
2
3
No Break
```

Iteration Statements (Loops)

2. For Loop continue

Nested For Loop

e.g.

```
for i in range(1,3):
  for j in range(1,11):
    k=i*j
    print (k, end=' ')
  print()
```

Output

1 2 3 4 5 6 7 8 9 10

2 4 6 8 10 12 14 16 18 20

Iteration Statements (Loops)

3. Jump Statements

Jump statements are used to transfer the program's control from one location to another. Means these are used to alter the flow of a loop like - to skip a part of a loop or terminate a loop

There are three types of jump statements used in python.

1.break
2.continue
3.pass

Iteration Statements (Loops)

1.break

it is used to terminate the loop.

e.g.

```
for val in "string":
    if val == "i":
        break
    print(val)

print("The end")
```

Output

```
s
t
r
The end
```

Iteration Statements (Loops)

2.continue

It is used to skip all the remaining statements in the loop and move controls back to the top of the loop.

e.g.

```
for val in "init":
    if val == "i":
        continue
    print(val)
print("The end")
```

Output

```
n
t
The end
```

Iteration Statements (Loops)

3. pass Statement

This statement does nothing. It can be used when a statement is required syntactically but the program requires no action.

Use in loop

```
while True:
        pass  # Busy-wait for keyboard interrupt (Ctrl+C)
```

In function

It makes a controller to pass by without executing any code.

e.g.

```
def myfun():
    pass   #if we don’t use pass here then error message will be shown
print(‘my program')
```

OUTPUT

My program

Iteration Statements (Loops)

3. pass Statement continue

e.g.

```
for i in 'initial':
    if(i == 'i'):
        pass
    else:
        print(i)
```

OUTPUT

n
t
a
L

NOTE : continue forces the loop to start at the next iteration while pass means "there is no code to execute here" and will continue through the remainder or the loop body.